FORGE
BY THE FOUNDRY

©2008 The Foundry Visionmongers Ltd. All rights reserved.
Forge 2.0v1 User Guide
This manual, as well as the software described in it, is furnished under license and may only be used or copied in accordance with the terms of such license. This manual is provided for informational use only and is subject to change without notice. The Foundry assumes no responsibility or liability for any errors of inaccuracies that may appear in this book.

No part of this manual may be reproduced, stored in a retrieval system, or transmitted in any form without the prior written permission of The Foundry.

The Foundry logo is a trademark of The Foundry Visionmongers Ltd. All other products or brands are trademarks or registered trademarks of their respective companies or organisations.

Software engineering Bruno Nicoletti, Mailys Levassort.

Product testing Sean Brice, Jack Binks.

Writing and layout design Jonathan Barson using Adobe FrameMaker.

Proof reading Jonathan Barson.

Rev: 13 August 2008

Contents

Introduction

Installation 5
Licensing 6
Support 7
Other Foundry Products 7

Forge

Quick Start 9
Introduction 10
Usage 11
Image File Types 13
Examples 14
Tuning Forge 20
XML Parameters 21
FAQs 25

Tutorial

Images 27
Exercise 1 28
Exercise 2 32
Exercise 3 38
Exercise 4 39

Appendix A

Release Notes 45

Index

A-Z 51

INTRODUCTION

Welcome to this user guide for Forge. Forge is a standalone command line application that performs automated dirt detection and removal on a sequence of images.

Installation

Forge is available as a software download from our web site. The downloads are in tar format (tgz) for Linux and dmg format for Mac OS X. Forge is licensed using FLEXlm.

Forge on Linux

Follow these instructions.

1. Download Forge_2.0v1-linux-x86-release-32.tgz from our web site. (Or choose the 64bit version)
2. Move the file to a directory of your choice on a Linux machine. We recommend /usr/local/foundry
3. Extract the files from the archive using this command:
```
tar xvzf Forge_2.0v1-linux-x86-release-32.tgz
```
4. This creates a subdirectory called Forge_2.0v1-linux-x86-release-32 into which the command line application, forge, is copied.
5. To run Forge "cd" to this directory and type:
```
./bin/forge
```

Forge is installed but not licensed.

Forge on Mac OS X

Follow these instructions.

1. Download Forge_2.0v1-mac-universal-release-32.dmg from our web site.
2. Double click on the dmg file to create a pkg file.

3. Double click on the pkg file and follow the on-screen instructions.

Forge is installed but not licensed. You can find forge in `/usr/local/foundry/Forge_2.0/`

Licensing

Forge uses FLEXlm license key encryption. All licenses are floating. This allows you to run Forge from any networked machine. Licenses are taken from a license server. Setting up a license server is beyond the scope of this document but is fully covered in the Foundry FLEXlm Tools User Guide (FFT) which is available to download from our web site. However, it's worth giving a quick overview of the steps involved in setting up a license server and getting a client machine to grab a license.

Note

For each forge that is run, a license will be requested from the license server. When forge completes the dust busting, it will give the license back. On a dual proc machine, if you run a forge on each cpu simultaneously then two licenses will be taken from the license server.

Overview

First you need to configure your license server. Pick a machine that is on all the time. Install the Foundry FLEXlm Tools software. You can download this from our web site.

The license key goes in any plain text file with a .lic file extension in the following directory. Don't save the key in a rich text (RTF) file as it won't work. This recommended location varies depending on the operating system you are using, and is as follows:

On Mac OS X:
`/Library/Application Support/TheFoundry/FLEXlm/`

On Linux:
`/usr/local/foundry/FLEXlm/`

You should then start the license server. Please refer to the FFT User Guide. At this point the license server should be managing the floating forge licenses.

Next you need to tell each networked machine running Forge where to get the license key from. There are several ways of doing this but the easiest is by pointing the environment variable FOUNDRY_LICENSE_FILE at the license server. For example:

```
setenv FOUNDRY_LICENSE_FILE @fred
```

Further Information

Further information on licensing Forge, setting up a floating license server, adding new license keys and managing license usage across a network you should read the Foundry FLEXlm Tools (FFT) User Guide which can be downloaded from our web site.

Support

If you have trouble installing, licensing or running Forge please contact our support team support@thefoundry.co.uk

Other Foundry Products

The Foundry is a leading developer of plug-in visual effects for film and video post production. Its products include Tinder, Tinderbox, Keylight and Anvil and run on a variety of compositing platforms including Flame, Flint, Fire, Inferno

and Smoke from Discreet, Shake, Avid DS and After Effects. For the full list of products and supported platforms see our web site www.thefoundry.co.uk

Nuke is an Academy award winning compositor. It has been used to create extraordinary images on scores of feature films including Flags of our Fathers, King Kong and The Day After Tomorrow.

Tinder plug-ins have been developed by the Foundry and have been sold to hundreds of users throughout the world. They are available on the leading compositing platforms including Autodesk Flame, Avid DS and Adobe After Effects.

Furnace is a rich collection of image processing tools to help compositors tackle common problems when working on films. Plug-ins include wire removal, retimer, rig removal, texture plug-ins, grain tools, steadiness and deflicker.

Anvil is a set of colour correction and colour manipulation tools. It was originally developed by Paul Grace at First Art and is now available on a variety of platforms.

Keylight is an award winning blue/green screen keyer giving results that look photographed not composited. The Keylight algorithm was developed by the Computer Film Company who were honoured with a technical achievement award for digital compositing from the Academy of Motion Picture Arts and Sciences.

Visit The Foundry's web site at www.thefoundry.co.uk for further details.

FORGE

Forge is a standalone command line application from The Foundry that performs automated dirt detection and removal of that dirt on a sequence of images. No operator assistance is required after submitting the command. Figure 1 shows a

Figure 1. Scanned Image.

Figure 2. Detected Dirt.

scanned image with flecks of dirt. They are a bit hard to see, you'll have to take my word for it. Figure 2 is an exaggerated matte showing where dirt has been detected by Forge.

Rather than perform its own dirt detection, it can also take the output of an infrared pass from a Northlight film scanner (embedded in the dpx files) and use this to clean the clip. If your scanner is not a Northlight but capable of producing IR passes you may be able to extact and convert this to a format that Forge can read.

Quick Start

If you can't wait to have a go, and you have 10 frames knocking around that you want cleaning, try typing this command in a shell. It creates 10 clean frames in the current directory from 10 dirty ones:

```
> forge dirty.#.cin clean.#.cin -t 1-10
```

Not clean enough? Try this command:

```
> forge dirty.#.cin clean.#.cin -t 1-10 -a 5
```

Introduction

Forge's dirt removal will automatically detect and remove specs of dust and dirt from a frame. It identifies dirt by looking for objects that appear only in one frame, but not in the surrounding frames. Motion compensation is used to make sure moving objects are not falsely identified as dirt to be removed.

For example:

- A spec of dirt that appears for only one frame will be classified as dirt.
- A football being kicked across the image will not be classified as dirt because, after taking account of motion, it appears in each frame of the sequence.
- A vertical scratch in a sequence will not be classified as dirt as it appears in the same place in each frame.
- Dirt on the camera lens or in the telecine gate will not be classified as dirt as it appears in the same place on each frame.

Having detected the dirt, the algorithm produces a seamless repair by taking motion compensated pixels from the surrounding frames and interpolating them into the dirt region.

The main control provided by to forge is via the '-a [n]' command line option (autodetect) where the agression factor n=1,2,3,4 or 5. For example:

```
> forge dirty.#.cin clean.#.cin -t 1-10 -a 5
```

This controls the trade off between falsely identifying real

features as dirt and failing to spot actual dirt. Often, even if a region of image has been falsely detected as dirt, it will be repaired perfectly as the dirt will not have corrupted the motion in the region, allowing a high quality motion compensated repair. '-a' is an overall tuning control that sets ten parameter values. For the advanced user it is also possible to fine tune these individual parameters for better results on specific sequences. See "Tuning Forge" on page 20.

Forge can also be used to perform just the motion compensated repair by feeding in separate 'dirt' masks. This sequence of files is used to indicate where dirt has previously been detected, for example from a previous run of forge, or the embedded infrared pass from a Northlight film scanner. If from an IR pass, it will speed up Forge as it does not need to perform the dirt detection phase, and lead to a better repair as no area will incorrectly identified as dirt (which can happen with strobing lights).

Usage

Usage information is printed on the command line. Just type forge to display this. Note that the dirt removed output will only have RGB channels. Any embedded alpha in the source image will be stripped in the output image.

Usage: forge inFileSpec outFileSpec [options]

where:

- **inFileSpec** - is a specification for the files to use as input (eg 'fred.#.dpx'). Supported file formats are Cineon, DPX and SGI.

- **outFileSpec** - is a file specification for the cleaned up output.

Options for controlling the dirt removal

- **-a arg** - sets the aggression of the dirt autodetection, where arg=1 is the minimum and arg=5 is the maximum.
- **-p arg** - where arg is an XML file that contains settings for the dirt removal. These will override the settings corresponding to the current autodetection level (specified with --autodetect).
- **--dumpparams** - prints an example XML file with the current settings. Use this as a starting point for your own settings.
- **-P arg** - specifies an individual dirt removal parameter where arg is NAME=VALUE.

Processing options

- **-t arg** - Specifies the frame or frames to process. Arg can be a single frame or a range of frames. You can process every other frame.

  -t 5 renders frame 5.

  -t 1-10 renders frames 1,2,3,4,5,6,7,8,9,10.

  -t 1-10,2 renders frames 1,3,5,7,9.

  -t 1,10,2 is the same as -t 1-10,2
- **-m arg** - use this flag for mutiprocessing where arg is the number of cpus you want to use. This option will spawn a number of forges and divide the frame range up amonst them. Each forge takes a license from the floating license server.

Predetected dirt options

- **-i arg** - an optional sequence of files that indicate which regions are dirt and should be repaired. Setting this turns off autodetection.
- **--indirtchannel arg** - sets the components to use as the dirt signal from the indirt image.

  --indirtchannel RGB (luminance)

  --indirtchannel A (alpha channel)

  --indirtchannel P (padding bits from a 10bit DPX file)

  --indirtchannel N (Northlight scanner DPX file)

  --indirtchannel auto (it makes a guess!)

Output image options

- **-o arg** - sets the filename of an optional sequence of frames that show where the dirt was detected.
- **-outdirtchannel arg** - sets the channels to write to.

  --outdirtchannel RGB

  --outdirtchannel A (default)
- **--outdirtinalpha** - writes the dirt signal into an alpha channel of the cleaned output image.

Miscellaneous options

- **-h** - prints the help message.
- **-v** - prints the version number of forge.
- **-q** - quiet mode. Nothing is printed to the shell.

Image File Types

SGI, DPX and Cineon files are currently supported. Forge automatically recognises the type of an input file, regardless of the file suffix. If the input file type cannot be recognised, the program will raise and error and exit.

To specify the output file type, use a suffix on the file name. For example:

- .dpx for DPX format files.
- .cin for Cineon format files.
- .sgi for Silicon Graphics format files.

Examples

Here are some examples. Please note that all commands are prefixed by a > symbol to denote the command prompt in the shell. Also any output from the forge command has been ignored in these examples.

```
> cd /pics/scans
> ls
dirty.0001.rgb
dirty.0002.rgb
dirty.0003.rgb
dirty.0004.rgb
dirty.0005.rgb
>
```

File Conversion

To convert the SGI files to DPX format during the cleaning:

```
> forge /pics/scans/dirty.#.rgb clean.#.dpx -t 1-5
```

To convert the SGI files to Cineon format:

```
> forge dirty.#.rgb clean.#.cin -t 1-5
```

Pattern Matching

Use the '#' character to say where the numbers should be in a file name. Forge will automatically scan for files that match and figure out the appropriate number of zero padded digits

to use. Using the # notation in the output filename will give you the same number of zero padded digits as the input.

```
> forge dirty.#.rgb clean.#.rgb -t 1-5
> ls
clean.0001.rgb
clean.0002.rgb
clean.0003.rgb
clean.0004.rgb
clean.0005.rgb
>
```

You can use an '@' character to indicate exactly how many zero padded digits there are in the filename. You can even change the padding on the output.

```
> forge dirty.@@@@.rgb clean.@@@@.rgb -t 1-5
> ls
clean.0001.rgb
clean.0002.rgb
clean.0003.rgb
clean.0004.rgb
clean.0005.rgb
> forge dirty.@@@@.rgb clean.@@@@@@.rgb -t 1-5
> ls
clean.000001.rgb
clean.000002.rgb
clean.000003.rgb
clean.000004.rgb
clean.000005.rgb
> forge dirty.@@@@.rgb clean.@@.rgb -t 1-5
> ls
clean.01.rgb
clean.02.rgb
clean.03.rgb
clean.04.rgb
clean.05.rgb
>
```

You can process single frames with either of these commands:

```
> forge dirty.#.rgb clean.#.rgb -t 32
> forge dirty.#.rgb clean.#.rgb -t 32-32
> ls
clean.0032.rgb
>
```

Skipping Frames

To process every other frame:

```
> forge dirty.#.rgb clean.#.rgb -t 1-5,2
> ls
clean.0001.rgb
clean.0003.rgb
clean.0005.rgb
>
```

To process every third frame:

```
> forge dirty.#.rgb clean.#.rgb -t 1-5,3
> ls
clean.0001.rgb
clean.0004.rgb
>
```

This command is equivalent if you prefer commas to hyphens:

```
> forge dirty.#.rgb clean.#.rgb -t 1,5,3
> ls
clean.0001.rgb
clean.0004.rgb
>
```

Flag Ordering

You can put the flags before or after the input and output filenames. So these are equivalent.

```
> forge dirty.#.rgb clean.#.rgb -t 1-5
> forge -t 1-5 dirty.#.rgb clean.#.rgb
```

Multiprocessing

So, you've got more than one cpu and you want to put them all to work. Try this command if you have 2 cpus.

```
> forge dirty.#.rgb clean.#.rgb -t 1-5 -m 2
```

and if you have 4 cpus try this.

```
> forge dirty.#.rgb clean.#.rgb -t 1-5 -m 4
```

Note that running Forge on 2 cpus simultaneously will check out 2 licenses from the license server.

Note that if you use -m 4 on a machine that has 2 cpus it will use more memory and more licenses than using the correct -m flag. It may also take slightly longer to process, but the frames rendered will be correct. So it's best to get this number right.

Saving the Dirt and Cleaned Images

As well as processing and saving the cleaned images, you can also save a single component image of the dirt that was found in the original scans by using the -o flag.

```
> forge dirty.#.rgb clean.#.rgb -t 1-5 -o dirt.#.rgb
> ls
clean.0001.rgb
clean.0002.rgb
clean.0003.rgb
clean.0004.rgb
clean.0005.rgb
dirt.0001.rgb
dirt.0002.rgb
dirt.0003.rgb
dirt.0004.rgb
dirt.0005.rgb
>
```

Important!

Many compositors are unable to read single channel images.

So it's worth adding the -o flag to force an RGB image to be written out rather than the default single channel image.

```
> forge dirty.#.rgb clean.#.rgb -t 1-5 -o dirt.#.rgb --
      outdirtchannel RGB
```

The only real disadvantage is that the dirt mask output has a larger file size than its single channel alternative.

Saving the Dirt in the Cleaned Images

Rather than have a seperate file holding the detected dirt you can add it into the alpha channel of the cleaned image. Output to dpx as this can hold an alpha channel.

```
> forge dirty.#.rgb clean.#.dpx --outdirtinalpha -t 1-5
```

Cleaning

By default the scans are cleaned with an "aggression" setting of 3. Allowable values are between 0 and 5, although 0 gives no cleaning, and are specified with a "-a" flag.

Default cleaning (both these commands produce the same output)

```
> forge dirty.#.rgb clean.#.rgb -t 1-5 -a 3
> forge dirty.#.rgb clean.#.rgb -t 1-5
```

Heavy cleaning

```
> forge dirty.#.rgb clean.#.rgb -t 1-5 -a 5
```

Light cleaning

```
> forge dirty.#.rgb clean.#.rgb -t 1-5 -a 1
```

Fine Tuning

To fine tune the cleaning parameters try this:

```
> forge --dumpparams
[this gives us the names of the parameters we can edit]
> forge dirty.#.rgb clean.#.rgb -t 1-5 -P dilate=5
```

Custom Settings in an XML file

Try this:

```
> forge --dumpparams > mysettings.xml
> vi mysettings.xml
[edit some of the numbers]
> forge dirty.#.rgb clean.#.rgb -t 1-5 -p mysettings.xml
```

IR Scans

If you have an image that identifies the dirt, for example from a previous run of forge or the output of an infrared pass on a film scanner, you can get Forge to use this rather than generate its own. In this example, we have a sequence of images mask.#.rgb that are used to tell Forge where the dirt is.

```
> forge dirty.#.rgb clean.#.rgb -t 1-5 -i mask.#.rgb
> ls
clean.0001.rgb
clean.0002.rgb
clean.0003.rgb
clean.0004.rgb
clean.0005.rgb
>
```

Some scans, like those from a Northlight film scanner, can produce an embedded picture of the dirt (IR Scan) and this can be used as follows:

```
> forge dirty.#.dpx clean.#.dpx -t 1-5 -i dirty.#.dpx
```

Forge always looks for a Northlight dirt mask embedded in the dpx file and if none is found the alpha of this input is used. Or if we want to force Forge to look at the dirt from a Northlight scanner we'd use:

```
> forge dirty.#.dpx clean.#.dpx -t 1-5 -i dirty.#.dpx --
indirtchannel N
```

Note *When using the -i flag the pathological motion detection (which is on by default) is automatically switched off.*

Tuning Forge

Advanced use only.

In order to understand how to tune the parameters it is first necessary to understand a bit more about the algorithms involved. Forge's dirt removal relies heavily on motion estimation to both detect the dirt and repair the image. Where the motion is pathological, e.g. multiple objects moving fast in multiple directions, it is much harder to perform motion compensation. This means both the dirt detection and repair can fail. In order to improve results in these regions we have a pathological motion detector, switched on by default. This detector is designed to flag regions where we are unlikely to calculate the correct motion. In these regions, we detune the motion based dirt detector and add a spatial dirt detector. Only if both detectors flag dirt do we actually believe there to be dirt.

It should be noted that if a pre-detected dirt mask is passed into forge, then no dirt detection is performed at all, so that a significant proportion of the parameters described below are not relevant, as they are to do with tuning dirt detection, as opposed to repair.

The -a flag lets you set the amount of dirt removal in Forge. If you wish, you can have more control over the dirt removal, although this is not for the feint hearted.

The -a flag is used to set 10 hidden parameters that are used in the dirt detection. You can display these 10 parameters for various dirt settings with the -dumpParams flag.

```
> forge --dumpparams
><forge context='multiinput'>
  <params>
    <!-- Dirt parameters -->
```

```
    <bool name="pathMotion" value="1"/>
    <double name="motionThreshold" value="0.9"/>
    <double name="motionSmoothing" value="1.5"/>
    <double name="safetyFactor" value="5"/>
    <double name="medianThreshold" value="0.05"/>
    <double name="dirtRejectThreshold" value="0.05"/>
    <double name="dilate" value="1"/>
    <double name="changeThreshold" value="1"/>
    <int name="medianSize" value="11"/>
  </params>
</forge>
```

If you save these to an xml file and edit them, you can then pass these back into forge with the --params argument.

```
> forge --dumpparams > mysettings.xml
[edit the mysettings.xml file]
> forge dirty.#.dpx clean.#.dpx -t 1-5 -a 1 --params
      mysettings.xml
```

The underlying parameters are set by first looking at the '-a' command line option and setting the corresponding pre-chosen values, then by overriding those from any XML parameter file that may have been passed in.

XML Parameters

These parameters are described here.

pathMotion

Type
Boolean

Default
True

Description

Where pathological motion exists in the sequence it is likely that the motion based dirt detection scheme will fail. A pathological motion detector is required to flag these regions and allow a modified dirt detection scheme to be used.

The pathological motion detector works by dividing the image into blocks and looking at the consistency of the estimated motion for each block across five frames.

Once each individual block has been assigned pathological motion or not, a smoothing algorithm is applied to the block to generate the pathological motion matte for the image.

Enabling this flag turns on the detection of such region.

Note *However, turning pathological motion off will speed up processing by about 30% (at the cost of more mistakes).*

Example

To switch off pathological motion, use either of these commands:

```
> forge dirty.#.rgb clean.#.rgb -t 1-5 -P pathMotion=false
> forge dirty.#.rgb clean.#.rgb -t 1-5 --param
      pathMotion=false
```

motionThreshold

Type

double

Description

This threshold is the level below which we declare a block of image to be undergoing pathological motion. A lower

threshold will increase the amount of image flagged as pathological motion.

Only used if pathMotion is turned on.

motionSmoothing

Type
double

Description
This is the amount of smoothing applied to the blocks of image that have been detected as moving with pathological motion. More smoothing will increase the amount of image flagged as pathological motion.

Only used if pathMotion is turned on.

safetyFactor

Type
double

Description
In the regions of pathological motion we need to be more cautious about detecting dirt based on motion. So rather than using the detectionThreshold used in the other regions we scale it by safetyFactor to be extra cautious.

medianThreshold

Type
double

Description
After median filtering, pixels that differ by more than medianThreshold are flagged as dirt by the spatial detector.

dirtRejectThreshold

Type
double

Default
0.05

Description
This is the main threshold below which we set a pixel to be dirt and above which we assume it is an image feature.

dilate

Type
double

Default
1

Description
This is the amount by which the pixels detected as dirt are grown to ensure that the whole region of dirt is correctly detected. Occasionally we only detect the centre of a piece of dirt and so without dilation we would correct the interior but leave a halo of dirt. If this is the case increase dilate until the whole of the dirt is removed. Note that the pre-detected dirt mask will be dilated by this amount as well, not just the auto-detected regions.

changeThreshold

Type
double

Description
After repairing the dirt the repaired pixels are compared with the original pixels. If they differ by less than changeThreshold it is assumed that they were incorrectly flagged as dirt and replaced with the original pixels.

medianSize

Type
integer

Description
As well as using motion to detect dirt in regions of pathological motion we additionally add a spatial check. This is based on filtering the image with a median filter to remove objects below a certain size. MedianSize sets the size of this median filter. Making it too large will increase computation time and falsely detect image objects as dirt, too small and dirt will be missed.

FAQs

Here are some frequently asked questions from customers running forge.

1. Why won't forge read my dpx files?

   The DPX file format is pretty loose and it's common to have files that cannot be read across different applications. The only workaround is to convert to a format that can be read. You should also send us a frame.
2. Why won't Baselight read the dirt mask I created in Forge?

The most likely explanation is that your dirt mask is a single channel image file which is, at the time of writing, unsupported by Baselight and many other applications. Re-run forge using the -outdirtchannel RGB to force an RGB output for your dirt mask and then load that into Baselight.

3. Why are 4 licenses taken when I run Forge with -m 4?

   A Forge license is taken each time it is run. The -m 4 flag runs 4 copies of Forge simultaneously and 4 licenses are taken.

4. I run forge and get a crash saying "Process 1855 died with status 0 sig 1". How do I fix this?

   Check that you have permissions to write the output files.

TUTORIAL

This tutorial will guide you through a number of common dirt removal problems.

Images

You probably have your own footage on which to try Forge but if not, feel free to download the tutorial images from the Forge product page on our web site www.thefoundry.co.uk and use them in the exercises below.

Figure 3. Tutorial Footage - Mike.

Once downloaded, you should have 30 Cineon frames 16bit 730x576 called Mike.####.cin

```
> ls
Mike.0001.cin
Mike.0002.cin
Mike.0003.cin
...
...
Mike.0030.cin
>
```

We also supply 5 frames 2048x1556 sourced from a Northlight film scanner as dpx files with an embedded IR pass. See "Exercise 3" on page 38.

```
> ls
0030.dpx
0031.dpx
0032.dpx
...
...
0034.dpx
>
```

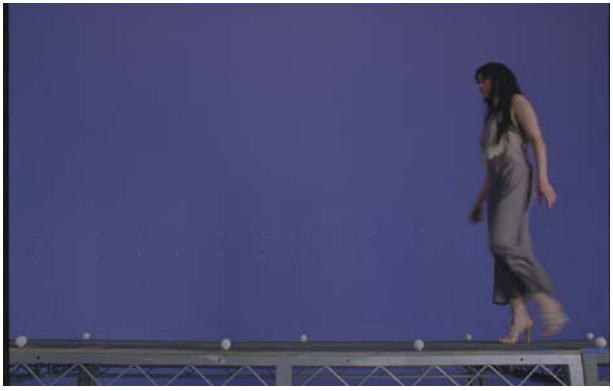

Figure 4. Tutorial Footage - Belle.

Exercise 1

In this exercise, we'll use Forge to clean the tutorial images as shown in Figure 3 and output both the clean result and a mask identifying the dirt. Try this command:

```
> forge Mike.#.cin clean.#.cin -o dirt.#.cin --
      outdirtchannel RGB -t 1-30
```

which will give the following output.

```
Processing from 1 to 30 with a frame increment of 1
```

```
Processing frame 1
  writing result to ./clean.0001.cin
  writing dirt mask to ./dirt.0001.cin
Processing frame 2
  writing result to ./clean.0002.cin
  writing dirt mask to ./dirt.0002.cin
Processing frame 3
  writing result to ./clean.0003.cin
  writing dirt mask to ./dirt.0003.cin
Processing frame 4
  writing result to ./clean.0004.cin
  writing dirt mask to ./dirt.0004.cin

etc
```

Using your favourite compositor, play through the original, clean and matte clips. If the cleaning is too harsh or too little modify the forge command with the -a [1-5] flag.

Multiprocessing

If you have a machine with 2 processors and you're in a hurry, try this command:

```
> forge Mike.#.cin clean.#.cin -o dirt.#.cin -t 1-30 -m 2
```

Here is a close-up of a hair on frame 18...

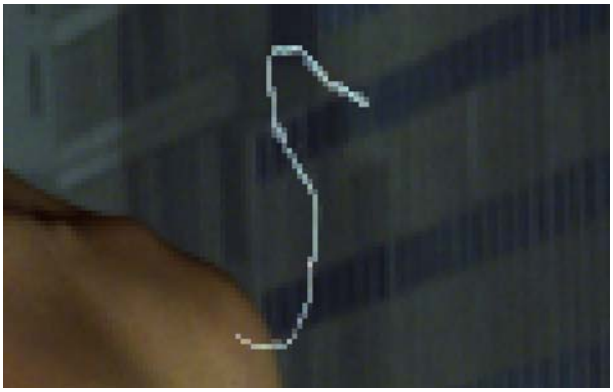

Figure 5. Original Frame.

...and this is the region that Forge detected to repair...

Figure 6. The Region to Repair.

...and the motion estimated repair itself.

Figure 7. Forge's Repair.

Exercise 2

In this exercise, we'll take our clean images, dirt mask and original images and restore through from the original any areas we think are image and not dirt. For the purposes of this exercise we're going to describe this process on several different compositors.

Shake

First up is Shake.

1. Start Shake and file in clean.#.cin, dirt.#.cin and Mike.#.cin.
2. Construct the tree as shown in Figure 8 on page 32. This composites the cleaned areas over the original footage. The single channel dirt mask is added as an alpha to the cleaned image using the Reorder node. This is then composited over the original Mike footage using the Over node. Insert a QuickPaint node after the dirt footage. We're going to paint this to bring through parts of the original image that we think should be there.

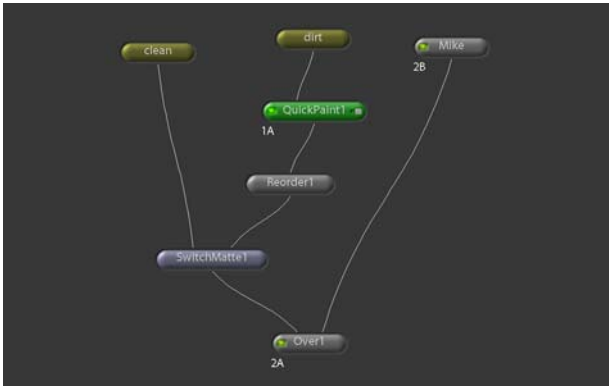

Figure 8. Shake Tree.

3. Step through each frame, painting areas of the dirt mask as appropriate. Figure 9 shows the Shake user interface. The dirt

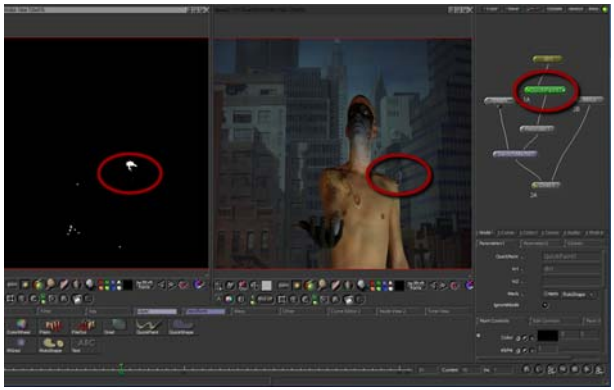

Figure 9. Shake User Interface.

mask that we're painting, the composite and QuickPaint node are highlighted.

4. Render the clip.

As an exercise for the reader, rather than assuming the repair is good and comping back areas that have been overcooked by forge, you could take the original footage and paint through only the forge repairs you want.

After Effects

Next is After Effects.

1. Start After Effects and load in the original Mike footage, the cleaned footage and the dirt mask.

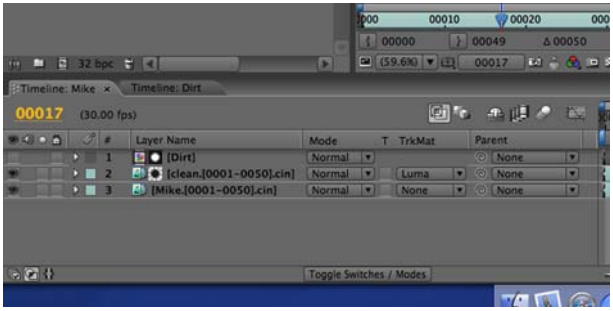

Figure 10. After Effects Timeline.

2. Layer the cleaned footage over the original footage in a composition and track matte the precomposed dirt layer to the cleaned top layer.

3. Add new viewers so that you can paint on the dirt layer while viewing the composition.

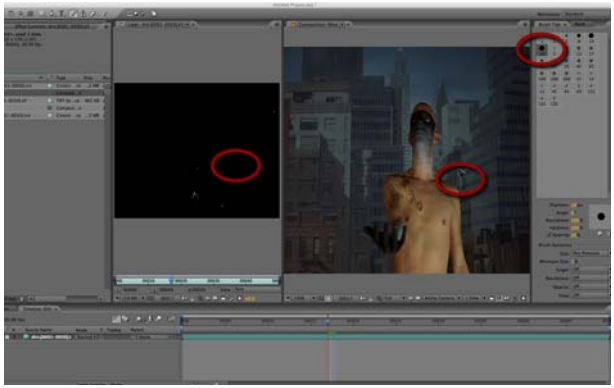

Figure 11. AE User Interface Showing 2 Viewers.

4. Paint on the precomposed dirt layer using the paint or vector paint tools. A large brush with full opacity black will enable you to quickly take out areas that were incorrectly repaired by Forge.

Baselight

Finally, let's use Baselight 3.1.

1. Start baselight. In a new job and new scene import the original, clean and dirt images.

2. Layer the original above the clean above the mask.

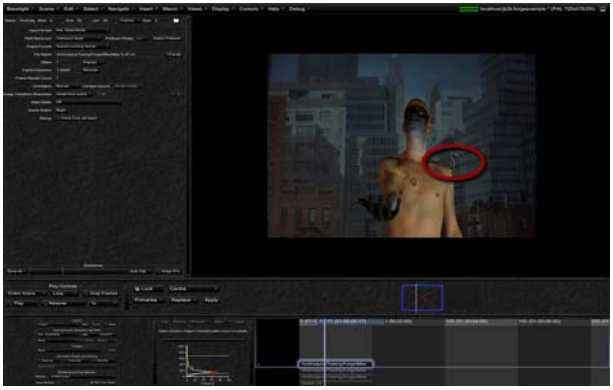

Figure 12. Baselight User Interface highlighting dirt.

3. Apply the Switch Dust filter to bring the clean footage through the original dirty footage.

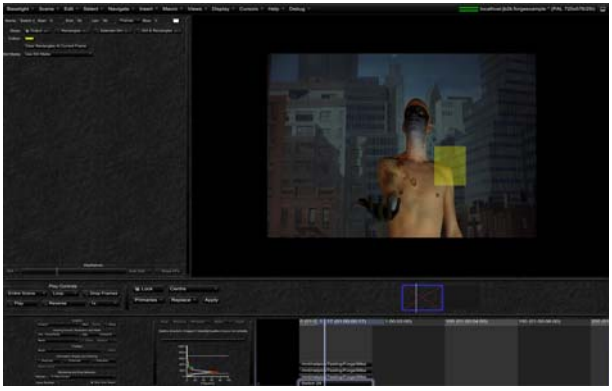

Figure 13. Switch Dust filter in yellow.

The user drags out rectangular area, shown in yellow. Note that only the pixels defined by the mask will be restored through and not the whole of the rectangular area.

4. Repeat this process stepping through the 30 frames in the clip.

If you want the clean footage by default and occasionally bring back the original footage using the Switch Dust filter then you should layer the clean footage at the top of the layer stack, then have the dirty original then the mask at the bottom.

Baselight has a convenience macro called the Dustbusting Approval Macro that sets up a stack for you, as long as your footage conforms to a naming convention. If you prefer, use this method. For more information please see the Baselight User Guide.

One more thing. If you have forgotten to render the dirt mask you can continue to use this method with just the clean and dirty frames. When restoring through using the yellow rectangle the whole of the pixels contained within the rectangle will be restored.

Exercise 3

Let's imagine you have some scanned originals from a Northlight scanner with an embedded dirt mask from an IR (infrared) pass. Download the example images from our web site. Use the following command to repair the scanned images.

```
> forge -t 30-34 -i #.dpx #.dpx clean.#.dpx
```

Exercise 4

Let's imagine you have four machines which are generally unused overnight and you wish to distribute your forge dust busting over these. You will use a fifth machine to submit the forge render jobs.

Now there are many ways of doing this. You may well have a system set up already to distribute Maya or Shake renders. You may be using commercial software to distribute renders. But assuming you haven't any of this, I hope this exercise will get you a foot on the ladder.

The exact details are beyond the scope of this document, but I will describe the principles involved in setting up a distributed render from a single script.

Assumptions

Let's assume all four machines are linux. There is also another linux machine that submits the forge render jobs. The machines are called:

1. red (2 cpus)
2. green (2 cpus)
3. blue (4 cpus)
4. yellow (2 cpus)
5. black (controller)

All machines have access to a shared disk which contains the images to be processed (100 dpx frames) and will store the cleaned images. The disk is called /pics. From any machine:

```
> cd /pics
> ls
0001.dpx
0002.dpx
0003.dpx
...
...
```

```
0100.dpx
```

Let's also assume we'll use a user login called **fred** on all machines.

ssh

You have to set up the machines so that you can run remote commands on the render machines without having to type passwords. You can do this with ssh. It's worth having a look at the man pages for this. It's also worth discussing the security implications with your Systems Administrator.

On **black** logged in as fred create a public key.

```
black> mkdir -p ~/.ssh (if it doesn't already exist)
black> chmod 700 ~/.ssh
black> cd ~/.ssh
black> ssh-keygen -t dsa
Generating public/private dsa key pair.
Enter file in which to save the key (/Users/fred/.ssh/
      id_dsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /Users/fred/id_dsa.
Your public key has been saved in /Users/fred/id_dsa.pub.
The key fingerprint is:
8a:ba:a4:65:2a:2c:2c:55:de:36:42:86:39:64:a1:4b fred@black
```

Copy the public key to the remote host **red**.

```
black> scp -p id_dsa.pub fred@red:
Password: ********
```

Log into the remote host **red** and install the public key.

```
>$ ssh -l fred red
Password: ********
red> mkdir -p ~/.ssh (If it doesn't already exist)
red> chmod 700 ~/.ssh
red> cat id_dsa.pub >> ~/.ssh/authorized_keys  (Appending)
red> chmod 600 ~/.ssh/authorized_keys
red> mv id_dsa.pub ~/.ssh (Optional, tidying up)
```

```
red> logout
```

Log back into red. You should be able to get in without a password.

```
black> ssh red
```

If that works you need to copy the authorized_keys to each of the other machines.

Software

You need to install forge on each remote render machines, or in a shared applications directory (e.g., /software). This amounts to untarring the distribution to a directory.

Licensing

You have to make sure that all machines can see the Foundry FLEXlm license server. This is best achieved by putting an environment variable in fred's **.cshrc** file in his home directory. When the job is run on the remote machine this will be picked up. You should also set up an alias to point to forge.

```
setenv FOUNDRY_LICENSE_FILE 30001@black
alias forge /software/forge
```

30001 is the port number and black is the name of the machine that is running the Foundry FLEXlm Server (lmgrd daemon).

Testing

It's worth testing forge over the ssh connection to red with this command.

```
> ssh red "forge -t 1-5 /pics/\#.dpx /pics/clean.\#.dpx"
```

The \ symbol before the # is used to "escape" this character.

Script

You need to create a script called render.csh that will execute the forge commands. On black:

```
> cat render.csh
ssh red "forge -t 1-100,4 -m 2 /pics/\#.dpx /pics/
      clean.\#.dpx" &
ssh green "forge -t 2-100,4 -m 2 /pics/\#.dpx /pics/
      clean.\#.dpx" &
ssh blue "forge -t 3-100,4 -m 4 /pics/\#.dpx /pics/
      clean.\#.dpx" &
ssh yellow "forge -t 4-100,4 -m 2 /pics/\#.dpx /pics/
      clean.\#.dpx" &
```

Note that this script will request 10 licenses from the Foundry FLEXlm license server. One for each instance of forge run. Red renders every fourth frame with the -t 1-100,4 flag. Green renders every fourth frame starting one frame later, -t 2,100,4. The & command spawns this job off and returns to execute the next command in the list. Without this, the jobs would run sequentially rather defeating the purpose of a distributed render.

The output is 100 frames called /pics/clean.#.dpx

Executing the Script

To run the script on black type this command:

```
> csh render.csh
```

Load Balancing

The astute amongst you will note that no attempt at load balancing has been made in this script. If machine yellow is your slowest machine it would be better to get that to render fewer frames. Ideally you want all your render machines to finish at the same time.

Improvements

Feel free to develop the above script to take command line arguments that split the frames across available machines rather than having to edit the file each time.

APPENDIX A

Release Notes

This section describes the requirements, new features, improvements, fixed bugs and known bugs & workarounds for each release of the Forge.

Forge 2.0v1

This is a new release of Forge with new features and improvements to existing features.

Requirements

1. Linux Centos 4.5 and Mac OS X (Tiger and Leopard)
2. Foundry FLEXlm Tools (FFT 4.0v1 or later).

Release Date

August 2008

New Features

1. Support for 32 and 64 bit.
2. Support for Mac Intel.
3. Support for Arri and Ditto scanners' dirt masks.
4. Support for 4K scans.

Improvements

1. Better command line options.
2. Up to a third faster than the previous version.

Fixed Bugs

1. Byte ordering (Bug ID 3566) for output dpx files was set to Big Endian and would fail to load in After Effects. This has been changed so that we only write out little endian files which are supported more widely including AFter Effects.
2. pathMotion (Bug ID 4463) is now set to on by default unless you are using an IR Scan to identify the dirt.
3. -a 0 flag (Bug ID 3496) now switches off dirt detection and copies the source to the output.

Known Bugs & Workarounds

There are no known bugs.

Forge 1.1v3

This is a maintenance release of Forge.

Requirements

1. Linux RH9 and RHEL3 and Mac OS X PPC.
2. Foundry FLEXlm Tools (FFT 4.0v1 or later).

Release Date

November 2006.

New Features

There are no new features.

Improvements

1. This pdf User Guide replaces the plain text guide in previous versions. A more extensive tutorial has been included.

Fixed Bugs

1. Spelling of pathological in the docs and application has been corrected.

Known Bugs & Workarounds

There are no known bugs.

Forge 1.1v2

This is a maintenance release of Forge.

Requirements

1. Linux
2. Foundry FLEXlm Tools (FFT 4.0v1 or later).

Release Date

August 2006.

New Features

There are no new features.

Improvements

There are no improvements.

Fixed Bugs

There are no fixed bugs.

Known Bugs & Workarounds

There are no known bugs.

Forge 1.1v2b1

This is a beta release of Forge.

Requirements

Linux

Improvements

1. Added multi-processing support by running copies of Forge on each of the available processors.

Fixed Bugs

1. If the output directory did not exist, Forge would fail. This has been fixed by creating the directory (and warning the user).
2. worked around a problem when cloning meta data from source files with dodgy DPX headers, such files now do not have their meta data cloned.

Forge 1.1v1

This is a maintenence release of Forge.

Requirements

Linux

Forge 1.1b1

This is a beta release of Forge.

Requirements

Linux

Improvements

1. Added support for FilmLight's IR dirt detection DPX file format.

Forge 1.0v1

This is the first release of Forge.

Requirements

Linux

Forge 1.0b9

This is a beta release of Forge.

Requirements

Linux

Fixed Bugs

1. The %06d style file specs would fail. This has been fixed.
2. Forge failed to run on some linux architectures. This has been fixed by compiling with -sse rather than -sse2.

Forge 1.0b8

This is a beta release of Forge.

Requirements

Linux

Improvements

1. Changed the file format specification to support '#' and '@'.
2. Added dirtmask input.
3. Changed it so that pathological motion detection is off by default. '-noPathMotion' option removed and '-pathMotion' added to turn on pathological motion detection on the command line.
4. If not performing pathological motion detection, memory demands are reduced by two full frames.
5. Dilation applied to optional dirtMask input.

INDEX

A-Z

Symbols
15
@ 15

A
After Effects 34
Anvil 7

B
Baselight 25, 35

C
cpus 17

D
Distributed rendering 39

F
FAQs 25
film scanner 19
FLEXlm 6
Foundry FLEXlm directory 6

I
Image File Types 13
infrared 38
infrared pass 19
Installation 5
IR pass 19
IR Scans 19

K
Keylight 7

L
Licensing 6

M
Multiprocessing 17, 29

N
Northlight 19, 38

P
pathalogical motion 22
Pattern Matching 14

R
Release Notes 45

S
Shake 32
single channel images 17
Skipping Frames 16
ssh 40
Support 7

T
Tinderbox 7
Tutorial 27

U
Usage 11

W
web site 8
www.thefoundry.co.uk 8

X
xml file 21